باسمه تعالی

گزارش پروژه

درس سیستم های عامل

استاد درس: جناب آقای دکتر رضا انتظاری ملکی

استاد حل تمرین: تعدادی از دانشجویان محترم دانشکده کامپیوتر

عرشیا گشتاسبی (۹۸۳۰۰۰۷۵)

امیرحسین صفایی مهر (۹۸۳۰۰۰۵۶)

دانشجویان کهاد مهندسی کامپیوتر

نیم سال دوم سال تحصیلی ۱۴۰۱-۱۴۰۲

فهرست

۱-معرفی پروژه

هدف از این پروژه طراحی و پیاده سازی ابزار انتقال فایل multi-threaded با استفاده از زبان برنامه نویسی C می باشد. این ابزار کاربران را قادر می سازد تا فایل ها را بین چندین سیستم در یک شبکه انتقال دهند، در حالی که از مفاهیم رشته ها، پردازش ها، فراخوانی های سیستمی، قفل ها و mutex استفاده می کنند. این پروژه برای انتقال فایل در یک سیستم لوکال پیاده سازی شده است.

این پروژه شامل موارد زیر می باشد:

- رابط کاربری

رابط کاربری به کاربر این امکان را می دهد که فایل ها را برای انتقال انتخاب کند و سیستم مقصد و دایرکتوری را برای انتقال مشخص کند.

نمونه نمونه در خط فرمان:

Multithreadedfiletransfer [source_path] [destination_address] [file splitting size]

در ادامه به جزییات این آرگومان ها اشاره خواهد شد و هر کدام توضیح خواهد داده شد.

- ماژول انتقال فایل

ماژول انتقال فایل وظیفه انتقال واقعی فایل ها بین سیستم مبدا و مقصد را بر عهده خواهد داشت. این ماژول از سوکت ها برای ارتباط بین سیستم ها استفاده می کند و از چند رشته برای انتقال چندین فایل به طور همزمان استفاده می کند.

این قسمت از کد با توجه به اطلاعات و دانش شبکه پیاده سازی شده است که به یک سرور و یک کلاینت نیاز داشتیم.

- مدیریت فرآیند

پـــروژه از فراخـــوانی هـــای سیســـتم fork() بـــرای ایجـــاد فرآینـــدهای فرزنـــد بـــرای مـــدیریت عملیـــات انتقال فایل استفاده می کند.

در دو قســـمت از fork اســـتفاده شـــده اســـت کـــه در ادامـــه بـــه آن در بخـــش هـــای بعـــدی اشـــاره خواهیم کرد.

- قفل ها و mutex

پـــروژه از قفـــل هـــا و mutex اســـتفاده مـــی کنـــد تـــا اطمینـــان حاصـــل شـــود کـــه چنـــدین رشـــته و فرآینـــد به طور همزمان به بخش های مهم کد دسترسی ندارند.

البتـــه در صـــورتی کـــه پیـــاده ســـازی درســـت و اصـــولی انجـــام شـــود مـــی تـــوان بـــدون نیـــاز بـــه ایـــن مـــوارد نیز کد را پیاده سازی کرد.

- اعلان خطا

پـــروژه دارای مکـــانیزم مـــدیریت خطـــای قـــوی خواهـــد بـــود کـــه کـــاربر را از هرگونـــه خطـــای رخ داده در حین عملیات انتقال فایل مطلع می کند.

۲-موارد مورد نیاز برای تحویل پروژه

این پروژه شامل موارد قابل تحویل زیر است:

۱ .کد منبع: کد منبع C برای ابزار انتقال فایل.

این مورد در ریپازیتوری در گیت به آدرس زیر قرار داده شده است، همچنین کد مربوط به سرور و کلاینت نیز به همراه پوش کردن کد ها در حین انجام پروژه موجود می باشد.

https://github.com/Arshiagosh/OSFinalProject402

۲ .راهنمای کاربر: راهنمای کاربر که نحوه استفاده از ابزار انتقال فایل را توضیح می دهد.

این مورد نیز در فایل final.c در ابتدای فایل به زبان انگلیسی تحت عنوان user manual قرار داده شده است.

۳ .ارائه: ارائه ای که پروژه و اجرای آن را توصیف می کند.

۴ .گزارش تست: یک گزارش آزمایشی که نتایج آزمایش ابزار انتقال فایل را با جزئیات نشان می دهد.

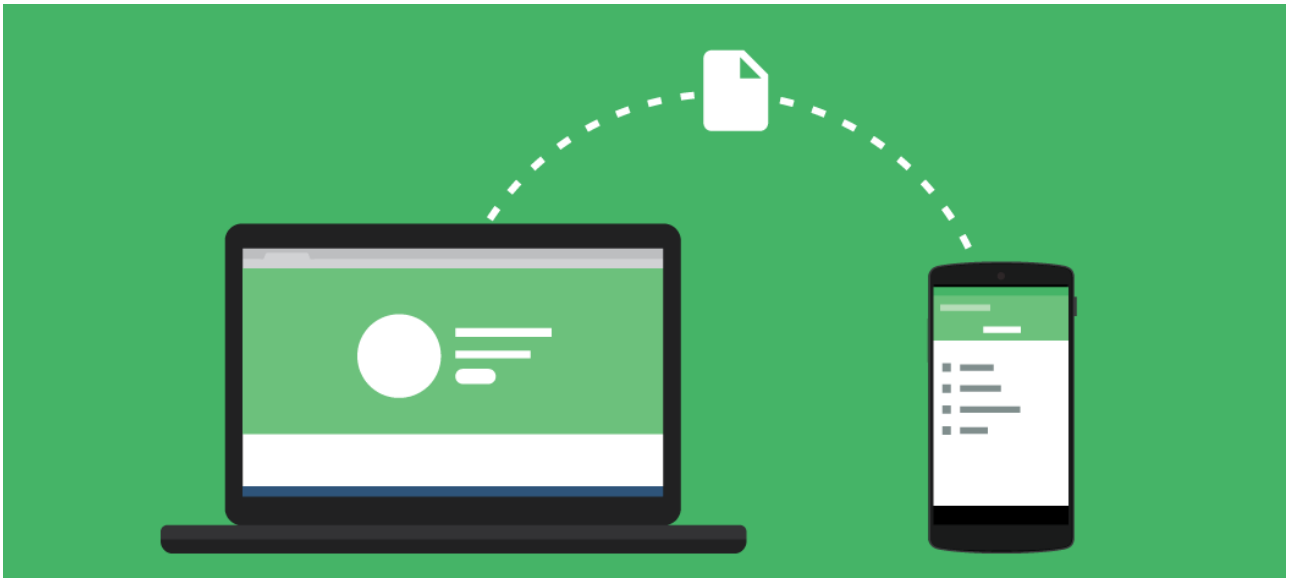
۵. گزارش پروژه: گزارش پروژه که جزئیات طراحی و اجرای ابزار انتقال فایل را نشان می دهد.

این مورد در حقیقت همین سندی است که شما در حال مطالعه آن می باشید.

۳-مقدمه

در ابتدا مروری بر روش های انتقال فایل از گذشته تا کنون داریم.

یک نمای کلی از تاریخچه انتقال فایل ها در سیستم عامل های مختلف بررسی شده است. در روزهای اولیه محاسبات، انتقال فایل معمولاً از طریق فلاپی دیسک یا سایر رسانه های ذخیره سازی قابل حمل انجام می شد. این دیسک ها به صورت فیزیکی از یک کامپیوتر به کامپیوتر دیگر منتقل می شوند تا فایل ها را انتقال دهند. با تکامل فناوری شبکه، انتقال فایل ها از طریق شبکه های محلی (LAN) با استفاده از پروتکل هایی مانند FTP (پروتکل انتقال فایل) و SMB (بلاک پیام سرور) امکان پذیر شد. این پروتکل‌ها به کاربران اجازه می‌دادند به فایل‌های ذخیره شده در رایانه‌های راه دور از طریق اتصال شبکه دسترسی پیدا کرده و انتقال دهند. در دهه ۱۹۹۰ و اوایل دهه ۲۰۰۰، ظهور اینترنت منجر به پذیرش گسترده ایمیل به عنوان وسیله ای برای انتقال فایل ها شد. کاربران می‌توانند فایل‌ها را به ایمیل‌ها ضمیمه کرده و آن‌ها را برای دیگران ارسال کنند، آنها می‌توانند پیوست‌ها را در رایانه‌های خود دانلود کنند. اخیراً سرویس‌های ذخیره‌سازی ابری مانندDropbox ، Google Driveو OneDriveبرای انتقال فایل‌ها محبوب شده‌اند. این سرویس‌ها به کاربران اجازه می‌دهند فایل‌ها را در فضای ابری ذخیره کنند و از هر دستگاهی که به اینترنت متصل است به آن‌ها دسترسی داشته باشند. امروزه اکثر سیستم عامل های مدرن دارای ابزارهای داخلی برای انتقال فایل ها بین دستگاه ها هستند. به عنوان مثال، ویندوز ۱۰ دارای قابلیتی به نام « Nearby sharing» است که به کاربران امکان می دهد با استفاده از بلوتوث یا وای فای به سرعت فایل ها را بین دستگاه های ویندوزی مجاور منتقل کنند. به طور مشابه، macOSشامل AirDropاست که به کاربران امکان می دهد به راحتی فایل ها را با سایر دستگاه های اپل در نزدیکی به اشتراک بگذارند. به طور کلی، تاریخچه انتقال فایل ها بین سیستم عامل ها با پیشرفت در فناوری شبکه، ظهور اینترنت و محبوبیت فزاینده خدمات ذخیره سازی ابری شکل گرفته است.



در ادامه به بررسی بعضی از مواردی که در پروژه شده است می پردازیم:

Thread و Process :Threadها فرآیندهای سبک وزنی هستند که امکان اجرای همزمان وظایف را در یک فرآیند فراهم می کنند. در این پروژه از multi-threading برای انتقال چند فایل به صورت همزمان استفاده خواهد شد. از سوی دیگر، فرآیندها واحدهای اجرایی مستقلی هستند که می توانند همزمان با سایر فرآیندها اجرا شوند. این پروژه از فراخوانی های سیستم fork() برای ایجاد فرآیندهای فرزند برای مدیریت عملیات انتقال فایل استفاده می کند.

Fork(): فراخوانی سیستم fork() برای ایجاد یک فرآیند جدید با کپی کردن فرآیند فراخوانی استفاده می شود. فرآیند جدید فرآیند فرزند و فرآیند اصلی فرآیند والد نامیده می شود. پردازش فرزند دارای شناسه فرآیند (PID) جدید است، اما فضای حافظه و توصیفگرهای فایل فرآیند والد را به ارث می برد. در این پروژه، fork() برای ایجاد فرآیندهای فرزند برای مدیریت عملیات انتقال فایل استفاده خواهد شد.

تمـــاس هـــای سیســـتمی: تمـــاس هـــای سیســـتمی تـــوابعی هســـتند کـــه توســـط سیســـتم عامـــل ارائـــه مـــی شـــوند و برنامـــه هـــای کـــاربر را قـــادر مـــی ســـازند تـــا بـــا سیســـتم تعامـــل داشـــته باشـــند. در ایـــن پـــروژه از فراخـــوانی هـــای سیســـتمی ماننـــد socket() بـــرای برقـــراری ارتبـــاط بـــین سیســـتم هـــا بـــرای انتقـــال فایـــل استفاده خواهد شد.

Locks و Mutex: از قفـــل هـــا و mutex بـــرای جلـــوگیری از دسترســـی همزمـــان بـــه منـــابع مشـــترک اســـتفاده مـــی شـــود. در ایـــن پـــروژه، از قفل‌هـــا و mutex اســـتفاده می‌شـــود تـــا اطمینـــان حاصـــل شـــود کـــه چنـــدین رشـــته و فرآینـــد بـــه طـــور همزمـــان بـــه بخش‌هـــای مهـــم کـــد دسترســـی ندارنـــد و در نتیجه از شرایط مسابقه و خرابی داده‌ها جلوگیری می‌شود.

۴-مروری بر کد

این یک برنامه C است که یک فایل یا یک دایرکتوری را به تکه های کوچکتر تقسیم می کند و آنها را از طریق شبکه به آدرس مقصد می فرستد. این برنامه از دو بخش تشکیل شده است - بخش سرور و بخش کلاینت. بخش سرور به اتصالات ورودی گوش می دهد و داده ها را از مشتریان دریافت می کند. بخش مشتری به سرور متصل می شود و داده ها را به آن ارسال می کند.

در ادامه به بررسی کد می پردازیم:

ابتدا کتابخانه های لازم شامل موارد زیر است:

```
28
29  #include <stdio.h>
30  #include <stdlib.h>
31  #include <string.h>
32  #include <unistd.h>
33  #include <arpa/inet.h>
34  #include <sys/wait.h>
35  #include <dirent.h>
36  #include <errno.h>
```

چند خط اول کد فایل های هدر هستند که شامل تعاریف برای توابع مختلف مورد استفاده در برنامه هستند.

این کتابخانه ها به ترتیب توابعی را برای ورودی/خروجی فایل، مدیریت حافظه، دستکاری رشته، شبکه، مدیریت فرآیند، پیمایش دایرکتوری و مدیریت خطا ارائه می دهند.

```
37
38  #define SIZE 1024
39  #define PORT 8080
40
```

در مرحله بعد، برنامه دو ثابت را تعریف می کند که بعداً در برنامه استفاده خواهند شد.:

SIZEنشان دهنده حداکثر اندازه بافر داده ای است که برنامه برای انتقال فایل ها استفاده می کند، در حالی که PORT نشان دهنده شماره پورتی است که سرور به آن گوش می دهد.

تابع main() نقطه ورودی برنامه است:

پارامتر argc تعداد آرگومان های ارسال شده به برنامه را نشان می دهد، در حالی که argv آرایه ای از رشته های حاوی آن آرگومان ها است.

این برنامه بررسی می‌کند که آیا تعداد آرگومان‌های صحیح ارائه شده است یا خیر:

```
40
41  int main(int argc, char *argv[])
42  {
43      if (argc != 4) {
44          fprintf(stderr, "Error: Invalid number of arguments provided.\n");
45          fprintf(stderr, "Usage: %s [source_path] [destination_address] [file splitting size]\n", argv[0]);
46          exit(EXIT_FAILURE);
47      }
48
```

این تابع اصلی است که در آن برنامه شروع به اجرا می کند. دو آرگومان نیاز دارد argc - و argv.آرگومان argc تعداد آرگومان های خط فرمان است که به برنامه ارسال می شود و argvآرایه ای از رشته های حاوی آن آرگومان ها است.

در این مورد، برنامه انتظار دارد که سه آرگومان ارسال شود – مسیر فایل یا دایرکتوری منبع، آدرس مقصد و اندازه تقسیم فایل. اگر این آرگومان ها ارائه نشود، برنامه یک پیغام خطا چاپ می کند و خارج می شود. در غیر این صورت، برنامه پیغام خطا را چاپ می کند و با وضعیت خرابی خارج می شود.سپس برنامه سعی می کند دایرکتوری مشخص شده توسط آرگومان اول را باز کند:

در صورت موفقیت، برنامه به دو فرآیند تقسیم می شود: یکی به عنوان سرور و دیگری به عنوان مشتری. فرآیند فرزند به سرور تبدیل می شود:

اگر مسیر منبع یک دایرکتوری باشد، برنامه ابتدا وجود دایرکتوری را بررسی می کند. اگر این کار را انجام دهد، یک بار برای ایجاد یک فرآیند سرور و سپس دوباره برای ایجاد فرآیند مشتری فورک می کند. کلاینت به سرور متصل می شود، مسیر دایرکتوری و اندازه تقسیم فایل را به سرور می فرستد، که به نوبه خود لیست فایل های موجود در دایرکتوری را ارسال می کند. سپس کلاینت هر فایل را به تکه هایی با اندازه مشخص تقسیم کرده و به

سرور ارسال می کند. سرور تکه ها را دریافت می کند، آنها را در فایل های مربوطه در فهرست مقصد می نویسد و سپس یک تاییدیه را برای مشتری ارسال می کند.

```
49      DIR* dir = opendir(argv[1]);
50      if (dir)
51      {
52          // Directory
53          printf("Source path is a directory\n");
54
55          int pid = fork();
56          if (pid == 0)
57          {
58
59              char cmd1[100];
60              sprintf(cmd1,"server/server %s %d %s",argv[2],PORT,argv[3]);
61              if(system(cmd1) == -1){
62                  fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd1);
63                  exit(EXIT_FAILURE);
64              }
65          }
66          else if (pid > 0)
67          {
68
69              wait(NULL);
70              char cmd2[100];
71              sprintf(cmd2,"client/client %s %d %s -d",argv[1],PORT,argv[3]); // -d flag for directory
72              if(system(cmd2) == -1){
73                  fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd2);
74                  exit(EXIT_FAILURE);
75               }
76          }
77          else
78          {
79              fprintf(stderr, "Error: Fork failed.\n");
80              exit(EXIT_FAILURE);
81          }
82      }
83      else if (ENOENT == errno)
84      {
85
```

اگر اولین آرگومان ارسال شده به برنامه یک دایرکتوری باشد، برنامه فرض می کند که کاربر می خواهد چندین فایل و دایرکتوری ارسال کند. در این مورد، برنامه به دو فرآیند تقسیم می شود - یکی برای سرور و دیگری برای مشتری.

فرآیند فرزند سرور را با اجرای سرور قابل اجرایی با آدرس مقصد و شماره پورت به عنوان آرگومان شروع می کند. اگر سرور شروع به کار نکند، پردازش فرزند پیام خطا را چاپ می کند و خارج می شود. فرآیند والد منتظر می ماند تا پردازش فرزند به پایان برسد.

پس از آن، فرآیند والد، کلاینت را با اجرای فایل اجرایی با مسیر منبع، شماره پورت، اندازه تقسیم فایل و پرچم -d به عنوان آرگومان شروع می کند. پرچم -d نشان می دهد که منبع

یک دایرکتوری است. اگر کلاینت شروع به کار نکند، فرآیند والد یک پیام خطا چاپ می کند و خارج می شود.

تابع printf() پیامی را چاپ می کند که نشان می دهد مسیر منبع یک دایرکتوری است.

سپس برنامه با استفاده از تابع fork(). در فرآیند فرزند (pid == 0)، برنامه یک فرمان پوسته برای راه اندازی سرور ایجاد می کند و آن را به تابع system() می دهد تا اجرا شود. تابع sprintf() برای ساخت رشته فرمان استفاده می شود که از آرگومان های زیر تشکیل شده است:

server/server: مسیری که به سرور قابل اجرا می رسد

argv[۲]: آدرس مقصد

PORT: شماره پورت

argv[۳]: اندازه تقسیم فایل

اگر تابع system() خطای (-۱) را برگرداند، برنامه پیغام خطا را چاپ می کند و با وضعیت خرابی خارج می شود.

در فرآیند والد (pid > 0)، برنامه با فراخوانی تابع wait() منتظر می ماند تا پردازش فرزند تکمیل شود:

پس از تکمیل فرآیند فرزند، برنامه دستور شل دیگری را برای راه اندازی کلاینت ایجاد می کند و آن را به تابع system() می دهد تا اجرا شود:

```
83      else if (ENOENT == errno)
84      {
85
86          printf("Error: Source path '%s' does not exist.\n", argv[1]);
87          exit(EXIT_FAILURE);
88      }
89      else
90      {
91          // file
92          printf("Source path is a file\n");
93
94          char cmd1[100];
95          sprintf(cmd1,"server/server %s %d %s",argv[2],PORT,argv[3]);
96          char cmd2[100];
97          sprintf(cmd2,"client/client %s %d %s",argv[1],PORT,argv[3]);
98
99          int pid = fork();
100         if (pid == 0)
101         {
102             wait(NULL);
103             if(system(cmd2) == -1){
104                 fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd2);
105                 exit(EXIT_FAILURE);
106             }
107         }
108         else if (pid > 0)
109         {
110             if(system(cmd1) == -1){
111                 fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd1);
112                 exit(EXIT_FAILURE);
113             }
114         }
115         else
116         {
117             fprintf(stderr, "Error: Fork failed.\n");
118             exit(EXIT_FAILURE);
119         }
```

تابع sprintf() دوباره برای ساخت رشته فرمان استفاده می شود که از آرگومان های زیر تشکیل شده است:

Client/Client: مسیری که به کلاینت قابل اجرا می رسد

argv[۱]: مسیر منبع

PORT: شماره پورت

argv[۳]: اندازه تقسیم فایل

-d: پرچمی که نشان می دهد مسیر منبع یک دایرکتوری است

اگر تابع system() خطای (-۱) را برگرداند، برنامه پیغام خطا را چاپ می کند و با وضعیت خرابی خارج می شود.

اگـــر برنامـــه نتوانـــد دایرکتـــوری مشـــخص شـــده (ENOENT) را بـــاز کنـــد، فـــرض مـــی کنـــد کـــه آرگومان اول یک فایل را نشان می دهد و بر این اساس ادامه می دهد:

```
91          // file
92          printf("Source path is a file\n");
93
94          char cmd1[100];
95          sprintf(cmd1,"server/server %s %d %s",argv[2],PORT,argv[3]);
96          char cmd2[100];
97          sprintf(cmd2,"client/client %s %d %s",argv[1],PORT,argv[3]);
98
99          int pid = fork();
100         if (pid == 0)
101         {
102             wait(NULL);
103             if(system(cmd2) == -1){
104                 fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd2);
105                 exit(EXIT_FAILURE);
106             }
107         }
108         else if (pid > 0)
109         {
110             if(system(cmd1) == -1){
111                 fprintf(stderr, "Error: Failed to execute system command '%s'\n", cmd1);
112                 exit(EXIT_FAILURE);
113             }
114         }
115         else
116         {
117             fprintf(stderr, "Error: Fork failed.\n");
118             exit(EXIT_FAILURE);
119         }
120     }
121
122     return 0;
123 }
```

اگـــر مســـیر منبـــع یـــک فایـــل باشـــد، برنامـــه دو بـــار فـــورک مـــی کنـــد تـــا یـــک فرآینـــد ســـرور و یـــک فرآینـــد مشـــتری ایجـــاد شـــود. ســـرور بـــه اتصـــالات ورودی از مشـــتری گـــوش مـــی دهـــد، فایـــل را از مشـــتری دریافـــت مـــی کنـــد و آن را در مقصـــد مشـــخص شـــده مـــی نویســـد. کلاینـــت بـــه ســـرور متصـــل مـــی شـــود، فایـــل را مـــی خوانـــد، آن را بـــه تکـــه هـــایی بـــا انـــدازه مشـــخص تقســـیم مـــی کنـــد، هـــر تکـــه را به سرور می فرستد و سپس اتصال را می بندد.

بـــه طـــور کلـــی، ایـــن برنامـــه یـــک راه ســـاده بـــرای انتقـــال فایـــل هـــا از طریـــق شـــبکه بـــا اســـتفاده از برنامـــه نویســـی ســـوکت در C ارائـــه مـــی دهـــد. بـــا ایـــن حـــال، دارای محـــدودیت هـــایی ماننـــد آســـیب پـــذیری های امنیتی احتمالی و ناتوانی در مدیریت کارآمد فایل های بزرگ است.

باتشکر از توجه شما